August 1990

Report No. STAN-CS-90-1325

Temporal Automata

by

Jean-François Lavignon and Yoav Shoham

Department of Computer Science

Stanford University

Stanford, California 94305

Temporal Automata*

Jean-François Lavignon† and Yoav Shoham
Robotics Laboratory
Computer Science Department
Stanford University
Stanford, CA 94305

August 29, 1990

Abstract

We present a new model of machines and their operation, called *temporal automata.* Characteristics of this model include explicit representation of process time, symmetric representation of a machine and of the enviroment in which it operates, the wiring together of asynchronous automata, and the ability to aggregate individual machines to form one machine at a coarser level of granularity. We present the mathematical theory of temporal automata, and provide examples of applying the model. We then relate temporal automata to traditional constructs such as finite automata and Turing machines, as well as to more recent formalisms such as statecharts and situated automata. Finally, we briefly describe a formal language for defining temporal automata, a compiler for that language, and a simulator for the output of that compiler.

*This work was partially supported by a grant from AFOSR
†Visitor from DRET, Direction des Recherches, Etudes et Techniques, France

1 Introduction

In this document we present a new model of machines and their operation. By a 'machine' we mean any structure participating in a process, be it electronic, mechanical or biological. Although we will often use the term 'computation,' it should be understood in the general sense of 'process.' Many models of computation already exist, and so we should first motivate the introduction of a new one.

1.1 Motivation and overview

The motivation behind our computational model lies in applications in which time plays a crucial role, and in which the environment is as important as the computing elements. In addition, our model is natural for applications that exhibit natural partitioning of computational elements, as well as a natural hierarchy on these computing elements. Two examples of such applications are robotics (and real-time systems in general), and Agent-Oriented Programming [17] (in which the computing elements are agents exchanging knowledge, beliefs and commitments).

We are aware of much past and present work that is closely related to our model. Nonetheless we believe that our model introduces some novel features, besides being natural and theoretically sound. It shares with most models some basic features, such as machines having state, input and output, and some notion of the current state and output depending on previous state and input. However, other features of our model distinguish it from previous models. Some recent models incorporate some of these features, but, to our knowledge, none incorporate all. The following intuitively-described features of our model are later developed rigorously.

- Each operation of the machine, be it rotating the camera by 30 degrees or taking a touch-sensor reading, requires time. Different operations in different machines require radically different times, leading to asynchrony. Temporal automata explicitly represent the duration of computation, allowing for different temporal models in different machines. Indeed, one of our initial motivations was reasoning about real-time systems in general, and robotics equipment in particular.

- In general, a machine alone does not define a computation; rather it is the machine coupled with its environment that together define the computation. Furthermore, just as the machine changes state, so does the environment. The changes in the environment and in the machine are mutually constraining. Only in the special case of *closed machines* is the computation independent of the environment.

- Like the machine, the environment too has structure, governing its state changes. In fact, the environment is simply another machine. In order for the machine and the

environment to affect one another, therefore, one must introduce the notion of the *wiring together* of machines.

(Although it will play no role in the formal development of the model, there is a conceptually-useful analogy to kinematics here. In the late 19th century, F. Reuleaux lay the foundation to modern kinematics by introducing the concept of the *kinematic pair* as the basic unit of analysis [13]. The simple underlying observation was that the possible motions of an object are only defined relative to another object with which it is in contact. Only relative to such a *joint* can the (rotational and translational) freedom of an object be specified. Furthermore, the object and the obstacle are completely symmetric, both being rigid 3D objects. By analogy, we can now speak about the *informatic pair*, a pair of machines, as the basic unit of analysis in computer science.)

- In fact, the machine-environment pair is only a special case of the wiring together of several temporal automata. In general we have a collection of machines that are wired together in a mutually constraining fashion. So we can easily model complex systems, each module of the system being a temporal automaton wired to the others.

  (This generalization is perfectly congruent with the analogy to kinematics, since, after introducing the basic kinematic pair, Reuleaux defined the more general *kinematic chain*; however in our context the word 'chain' would be misleading, suggesting a necessarily linear wiring).

- A collection of machines and a wiring among the machines together induce a new machine. Intuitively, the inputs (outputs) of the new machine are those of the individual machines that are not wired to an output (input) of a machine in the collection. (Note that since different machines may have different models of time, there will be some subtlety in defining the wiring together of arbitrary machines.) Thus one can reason about a collection of temporal automata as a single system. This introduces many different levels of granularity, and we can hierarchically decompose complex systems, starting with a 'black box' model and descending to as much detail as is desired.

1.2 Organization of the report

Sections 2, 3 and 4 define the model. Section 5 presents two examples of actual machines and their descriptions in our framework. Section 6 relates the model to the standard constructs of finite automata and Turing machines. Section 7 and 8 relate the model to more recent models that share some of the features of our model. Section 9 presents a formal language in which to define machines, and briefly describes a program which simulates the behavior of machines thus defined (a full description of the simulator is provided separately in [9]).

1.3 A sample automaton

Some of the folowing sections are rather abstract and it will help to couple them with a concrete example. We therefore now define, using the graphical conventions of finite-state automata, a particular standard deterministic finite automaton (or a Mealy machine), called M, to which we will make reference later in the text.

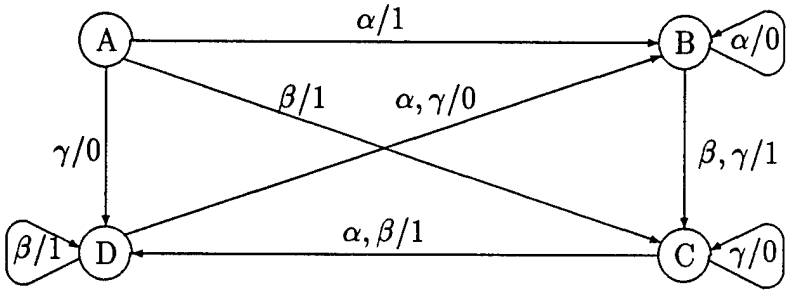


Figure 1 : The automaton M

The meaning of this representation is conventional; for example, if the M receives the signal α while in state A, it will emit the signal 1 and change to state B.

2 Preliminary definitions

2.1 Time structures

Any model of machines or computation must clearly define the notion of time, as a computation is a process which evolves over time (indeed, as was mentioned, we use the terms 'computation' and 'process' interchangeably). For example, the time implicit in finite state automata theory is a discrete succession of instants. These are the instants in which a new input signal is received, and based on it a new output signal is emitted and a state-change occurs.

We want to be able to model machines that can be asynchronous (not operating over the same periods and with events not happening at precisely the same time), and which can operate in continuous time or discrete time or a mix of continous and discrete time. In order to achieve this goal, we allow different components of a machine to be defined at different instants; thus we need to express the instants at which a component of a machine is defined. Such a set of instants will be called a 'time structure.' For example if we want to model a controller that takes a signal defined at any time and every second emits a command to an actuator, we need to talk about a continuous time for the input and a discrete time for the output. We will assume that each computation has a starting point. Thus our time structures will be finite in the past, and by convention we will take the initial point as 0.

The need for the last property in our definition of time structure will become clear when we introduce other concepts. In the following, R^+ stands for the non-negative reals and N for the non-negative integers.

Definition 1 : *A* time structure *is a set T included in or equal to R^+ such that $0 \in T$ and $\forall t \in R^+, sup\{x \in T \mid x \leq t\} \in T$.*

This last property means that T is closed on its right.

Examples :

- R^+
- N
- $N_\delta = \{ t \in R^+ \mid t = k * \delta \text{ for some } k \in N\}$

2.2 Entities

As was said in the introduction, a machine has inputs, internal states and outputs. All three have the common property of taking values over time. Since the machine may be asynchronous we need a time structure for each input, internal state or output. Similarly, inputs, internal states and outputs may differ on the domain from which they take their values. For example, usually the input of a digitizer takes its values over R and its outputs take their values over a discrete set. So each input, internal state and output must have a time structure and a domain of values. Below we define the neutral term *entity*, which may in fact be an input, an internal state or an output.[1]

Consider, for example, the automaton M defined in section 1.3. For M we can define three entities, an input entity i which takes its values from the set $\{\alpha, \beta, \gamma\}$, an internal state entity s which takes its values from the set $\{A, B, C, D\}$, and an output entity which takes its values from the set $\{0, 1\}$. If we assume that the input arrives at regular intervals we can take the time structure of the three entities to be equal to N.

An entity is formally define as follows.

Definition 2 : *An* entity *is a pair (T, D) where T is a time structure and D is a domain (set of values).*

Now that we have this notion of entity, we want to be able to speak of the value taken by this entity over time. The notion is clearly a function from T to D. We call it a *trace* of the entity.

[1] An entity can be thought of as a variable, something that takes different values over time; we did not use the term 'variable' since it is by now a loaded term.

Definition 3 : *A* trace *of an entity* $e = (T, D)$ *is a function* $v : T \rightarrow D$. *The collection of all functions* $T \rightarrow D$ *will be denoted by* V_e.

A trace is only defined on the time structure of an entity. Sometimes, however, we need at time t to know what is the last value an entity has taken. In order to do so, we must identify the latest time of T which precedes t and take the value of the entity at that time. For that we define the *continuous time extension* of the trace.

Definition 4 : *The continuous time extension of a trace* v *of entity* $e = (T, D)$ *is the function* $\overline{v} : R^+ \rightarrow D$ *defined by:*

$\overline{v}(t) = v(sup\{x \in T \mid x \leq t\})$

So if $t \in T$ (the time structure of e) then $\overline{v}(t) = v(t)$ and otherwise the value of $\overline{v}$ is the last value (in time) of v. Notice that by definition of time structures we have $sup\{x \in T \mid x \leq t\} \in T$, and therefore the continuous time extension of a trace is well defined.

2.3 Causal functions and transductions

We now want to define what it means for an entity e to be a function of other entities. Intuitively that means that the value taken by e is dependent on the values taken by the other entities.

For example, in the sample automaton M the output value at time t is defined if we know the value of the input and the internal state at time t. The output depends only on the current input and internal state and its value is related to the value of the input and internal state by the function defined by the table of figure 2.

	input		
state	α	β	γ
A	1	1	0
B	0	1	1
C	1	1	0
D	0	1	0

Figure 2

In this example the output depends only on the current values. This is not always the case. The outputs of a machine can depend at time t on all the inputs from 0 to t. So the idea of dependence of an entity e on a set of entities $I = \{i_1, i_2, ...\}$ is related to the existence of a function from $\Pi_{i \in I} V_i$ to V_e. The function associates to a trace for all the entities of I a trace for e.

We do not, however, allow any arbitrary functions. Intuitively, the temporal evolution of a machine is constrained by the principle of causality. The output at time t cannot depend on an input which will arrive after t. So we are interested in only a subset of the functions from $\Pi_{i\in I}V_i$ to V_e which we call *causal functions* and which are defined by:

Definition 5 : *A function f from $\Pi_{i\in I}V_i$ to V_e, where V_k denotes the set of function from the time structure T_k to the domain D_k, is a* causal function *iff:*
$\forall(v_{i_1}, v_{i_2}, ...), (v'_{i_1}, v'_{i_2}, ...) \in \Pi_{i\in I}V_i$, $\forall t \in T_e$
$[\forall i_k \in I,\ v_{i_k} = v'_{i_k}\ on\ T_{i_k} \cap [0,t]] \Rightarrow f(v_{i_1}, v_{i_2}, ...)(t) = f(v'_{i_1}, v'_{i_2}, ...)(t)$

This definition means that when an entity e depends on the set of entities I according to the causal function f, if you take two histories (i.e. a trace for each input which by f will define a trace for e) and if e takes different values at time t for those histories then the inputs must differ at some time before t in the two histories.

Causal functions are the most general concept that will support the definition of machines, but in fact they are too general to be of practical use. Instead we now introduce a subset of causal functions called *transductions.* The intuition behind the definition is that usually a part of a machine (an entity) takes as inputs the value of other parts of the machine, performs a computation for some amount of time (say δ), and outputs the value of the computation. This transduction is initiated at every instant of the entity's time structure. The computation can be characterized by a function from the domains of the inputs to the domain of the output and the delay δ. The following definition captures this intuition, as well as some information about the initialization of the entity and about the way to take the values of the inputs.

Definition 6 : *A* transduction *is a tuple $(e, I, \delta, f_{init}, f)$ where e is an entity (T_e, D_e), I is a set of entities, δ is a nonnegative real, f_{init} is a function from $T_e \cap [0,\delta)$ to D_e and f is a function from $\prod_{i\in I} D_i$ to D_e. The transduction defines a function F from $\Pi_{i\in I}V_i$ to V_e such that $\forall(v_{i_1}, v_{i_2}, ...) \in \Pi_{i\in I}V_i$:*

$$F(v_{i_1}, v_{i_2}, ...)(t) = \begin{cases} f_{init}(t) & \forall t \in T_e \mid t < \delta \\ f(\overline{v_{i_1}}(t-\delta), \overline{v_{i_2}}(t-\delta), ...) & \forall t \in T_e \mid t \geq \delta \end{cases}$$

In the above definition we provide a function f_{init} which defines the value of the entity before the entity has performed the first calculation. (Notice that if $\delta = 0$ then f_{init} plays no role, and if $T_e \cap [0,\delta) = \{0\}$ then only $f_{init}(0)$ plays a role.) Notice also the use of the continuous time extention function $\overline{v_i}$ rather than the trace v_i itself: an entity need not be synchronized with its inputs, that is, the instant at which the input's value is needed in order to determine the value of an entity need not lie in the time structure of the input. If it does not then the input value taken is the last value of the input available at the current time.

Terminology. We will say that a transduction $(e, I, \delta, f_{init}, f)$ is *from I to e.* If $\delta > 0$ then we will call the transduction a *durational transduction.*

We have the following property which relates transductions to causal functions:

Proposition 1 : *A transduction $(e, I, \delta, f_{init}, f)$ where either $e \notin I$ or $\delta \neq 0$ defines a causal function from $\Pi_{i \in I \setminus \{e\}} V_i$ to V_e*

(See appendix A for a proof of this proposition.)

Notice in particular that a durational transduction with feedback is still causal.

Example: Let us illustrate the concept of entity and transduction with a small example of an entity implementing a simple clock, a common device needed to implement features such as time out. The clock keeps track of the time elapsed since the emission of a signal. Specifically, the value of the clock is zero until it is activated, and from then on it increases in increments of ϵ, the clock grain size.

We first define the function f of two variables *start* and *clock*:

$$f(start, clock) = \begin{cases} 0 & \text{if } clock = 0 \text{ and } start = 0 \\ clock + \epsilon & \text{otherwise} \end{cases}$$

If the entity *start* emits the signal we want to keep track of, then the entity $clock = (N_\epsilon, R)$ (where $N_\epsilon = \{k * \epsilon \text{ for } k \in N\}$) and the transduction $(clock, \{start, clock\}, \epsilon, (0, 0), f)$ will implement a clock giving the amount of time elapsed since the emission of the signal with a precision of ϵ . We will have $v_{clock}(k * \epsilon) = f(\overline{v}_{start}((k-1) * \epsilon), v_{clock}((k-1) * \epsilon))$. So according to the definition of f, as soon as v_{start} becomes 1, v_{clock} will start to increase by ϵ every ϵ seconds. Then in every history the value of the entity ***clock*** can be use to implement time-out or other time-dependent behavior.

3 Causal systems and temporal automata

We have now all the elements needed to introduce our computational model. We first define the general notion of *causal systems.*

Definition 7 : *A* causal system *is a tuple (I, O, f) where I is a set of input entities, O is a set of output entities and f is a causal function from $\Pi_{i \in I} V_i$ to $\Pi_{o \in O} V_o$*[2].

[2] i.e. f is a family of causal functions $\{f_o\}_{o \in O}$ with $f_o : \Pi_{i \in I} V_i \rightarrow V_o$

The temporal evolution of a causal system depends on the traces of the input entities. The *environment* of a causal system determines the traces of the input entities and thus the history of this machine.

With each element $(v_{i_1}, v_{i_2}, ...)$ of $\Pi_{i \in I} V_i$ provided by the environment we have an *history* for the machine. For each history the value of the entity e at time $t \in T_e$ is $v_i(t)$ if $e = i \in I$ and $f_o(v_{i_1}, v_{i_2}, ...)(t)$ if $e = o \in O$.

As was said in the previous section, causal functions are somewhat too general to provide a useful model of computation. We therefore now define a more specific form of causal systems, *temporal automata.*

To define temporal automata we will use the notion of transductions which are special causal functions. But first we introduce some more notation.

Suppose we have a set E of entities and a set of transductions from a set included in E to some entities of E such that there is at most one transduction to each entity of E. We want to define the set of entities on which an entity e depends.

We introduce two relations $\mathcal{D}$ and $\mathcal{D}^0$. The semantics of $e\mathcal{D}a$ is that the transduction to e is from a set including a. The meaning of $e\mathcal{D}^0a$ is that the transduction to e is from a set including a and has a null delay. We denote by $\mathcal{D}(e)$ ($\mathcal{D}^0(e)$) the sets of entities standing in relation $\mathcal{D}$ ($\mathcal{D}^0$) with e.

Formally we define the relation $\mathcal{D}$ for each entity of E as follows: if there is a transduction to e, $(e, A, \delta, f_{init}, f)$ then $\mathcal{D}(e) = A$, otherwise $\mathcal{D}(e) = \emptyset$.

The definition of $\mathcal{D}^0$ is similar: if there is a transduction to e, $(e, A, 0, f_{init}, f)$ then $\mathcal{D}^0(e) = A$, otherwise $\mathcal{D}^0(e) = \emptyset$.

The transitive closure of those relations for the entity e will be denote by $\Delta(e)$ and $\Delta^0(e)$. If we have $\mathcal{D}_0(e) = \mathcal{D}(e)$ and the equation $\mathcal{D}_{n+1}(e) = \cup_{e' \in \mathcal{D}_n(e)} \mathcal{D}(e')$, then $\Delta(e) = \cup_{n \in N} \mathcal{D}_n(e)$. This set $\Delta(e)$ is exactly the set of the entities on which e depends. The definition of $\Delta^0(e)$ is similar but with the relation $\mathcal{D}^0$ instead of $\mathcal{D}$. The set $\Delta^0(e)$ is exactly the set of the entities on which e depends with a null delay.

We have now all the elements needed to define a *temporal automaton.*

Definition 8 *: An* temporal automaton *is a tuple $(I, S, O, \mathcal{T})$ where:*

- I is a set of input entities,
- S is a set of internal (state) entities,
- O is a set of output entities ,
- $\mathcal{T}$ is a set of transductions with the following properties:
$\forall e \in S \cup O$, $\mathcal{T}$ has a unique transduction to e and this transduction is from a set included in $I \cup S$
$\forall e \in S \cup O$, $\Delta(e)$ is finite and $e \notin \Delta^0(e)$

We will later relate this definition to the general definition of a causal system. Let us first illustrate this definition by encoding in it the automaton M defined in Section 1.3. As we said before, the automaton M has three entities:

- the input entity i, $(N, \{\alpha, \beta, \gamma\})$,
- the internal state entity s, $(N, \{A, B, C, D\})$
- the output entity o, $(N, \{0, 1\})$

The transduction for o is essentially defined by the function of Figure 2 whose name will be f. In addition we have $\delta = 0$ as in finite automata theory the output is simultaneous with the input. So the output transduction is $(o, \{i, s\}, 0, 0, f)$.

The transduction for s is essentially defined by the function g of the figure below. If we assume that initially s has value A , we have for f_{init} the function which associates A with time 0. The delay for s is 1 as in the finite automata theory an input produce a state transition by the time the next input arrives. So the transduction for s is $(s, \{i, s\}, 1, f_{init}, g)$.

	input		
state	α	β	γ
A	B	C	D
B	B	C	C
C	D	D	C
D	B	D	B

Figure 3

The temporal automaton evolves according the following equations:

$\forall t \in N \; \forall v_i \in \{\alpha, \beta, \gamma\}^N$

$v_o(t) = f(v_i(t), v_s(t))$

$v_s(t) = g(v_i(t-1), v_s(t-1))$ if $t > 0$

$v_s(0) = A$.□

In the definition of temporal automata, the property of the transductions assures that each internal or output entity does not depend on an infinity of entities and that there is no loop of zero delay transductions between internal entities. Note that the condition on Δ^0 always holds for an output entity o because $\Delta^0(o) \subset I \cup S$.

However, our definition of temporal automaton allows the transductions to be from S. This might seem to be a problem because the transductions to an entity of S use S itself. Actually we have the following proposition which guarantees that because there is no loop of non-durational transductions a temporal automaton is indeed a causal system.

Proposition 2 : *If M is a temporal automaton $(I, S, O, \mathcal{T})$ then there exists a unique causal function from $\Pi_{i \in I} V_i$ to $\Pi_{e \in S \cup O} V_e$ which satisfies the transductions equations.*

(See appendix A for a proof of this proposition).

Therefore a temporal automaton $(I, S, O, \mathcal{T})$ induces a causal system $(I, O, f_{\mathcal{T}})$ where $f_{\mathcal{T}}$ is the projection of the unique causal function from $\Pi_{i \in I} V_i$ to $\Pi_{e \in S \cup O} V_e$ which satisfies the equations of $\mathcal{T}$ on $\Pi_{o \in O} V_o$. We will call this function the function induced by the transductions of $\mathcal{T}$.

Before going on, let us define exactly when two systems (temporal automata or causal systems) are equivalent.

Definition 9 : *Two systems M and M' (M and M' can be either causal systems or temporal automata) are equivalent iff:*

- *$\exists g$ a one to one mapping from I' onto $I \mid \forall i' \in I'\ i' = g(i')$*
- *$\exists h$ a one to one mapping from O onto $O' \mid \forall o \in O\ o = h(o)$*
- *$\forall o \in O$, $\forall (v_{i_1}, v_{i_2}, ...) \in \Pi_{i \in I} V_i$,*
  $f'_{h(o)}(v_{g(i'_1)}, v_{g(i'_2)}, ...) = f_o(v_{i_1}, v_{i_2}, ...)$
  where f and f' denotes either the causal function (for a causal system) or the causal function induced by the transductions (for a temporal automaton) and f_o the projection of f on V_o.

4 Aggregating temporal automata

We now have a complete definition of temporal automata. We have been concerned with developing a model which is general, mathematically precise, and at the same time is a natural medium in which to encode actual machines. The mathematical precision will enable us later to compare our model to classical models of computation. However, alongside the theoretical issues, we are concerned with how readily actual machines can be encoded in our model. The definition of temporal automata was designed to correspond directly to our conceptions of process, and in the next section we provide two encodings of processes in actual machines. The further development of our model in this section does not increase its expressiveness, but does make it an even more practical tool for reasoning about machines and processes.

One property of real systems is that they are typically composed of multiple interconnected machines. For example a robot has sensors, actuators and control modules, each of which can be considered as a separate machine. The relation between these machines is that the outputs of some machines are used as the inputs of others. For example, the outputs of a sensor are used by a control module which transmits its commands to the actuators.

This ability to have a modular description of complex systems is essential if we want a model to be useful. In our framework each module of a system can be represented by a temporal automaton and the interactions between them are represented by the notion of a *connection.* A connection can be viewed as the wiring of the output of a machine to the input of another machine.

Definition 10 : *A* connection *from one entity e_1 to an entity e_2 is denoted $e_1 \rhd e_2$ or $e_2 \lhd e_1$ and is the causal function f from V_1 to V_2 which associates to $v_1 \in V_1$, $v_2 = f(v_1)$ defined by $\forall t \in T_{e_2}$, $v_2(t) = \overline{v_1}(t)$. (This definition implies that $D_{e_1} \subseteq D_{e_2}$.)*

Note that the time structures of e_1 and e_2 need not be equal; a connection does not imply a synchronisation between the two entities.

If we have several causal systems we can connect some output entities of some systems to the input entities of some others. We will speak about a *wiring* over a set of causal systems in that case.

Definition 11 : *Let $M_k = (I_k, O_k, f_k)$ be some causal systems. A* wiring *$\mathcal{W}$ over the M_k is a set of connections such that if $e \rhd e' \in \mathcal{W}$ then $\exists k \exists k' \mid e \in O_k$ and $e' \in I_{k'}$ and if $e_1 \rhd e' \in \mathcal{W}$ and $e_2 \rhd e' \in \mathcal{W}$ then $e_1 = e_2$.*

A wiring over a collection of causal systems defines a new system, which we will call *the system induced by the wiring.* Intuitively, the input entities of this system are all the inputs which are not involved in the wiring, and the output entities are all the outputs which are not involved in the wiring. Formally it is defined as follows.

Definition 12 : *Let $M_k = (I_k, O_k, f_k)$ be some causal systems and $\mathcal{W}$ a wiring over the M_k. The* system induced by the wiring *$\mathcal{W}$ is the system with input entities $\cup_k I_k \setminus \{e' \mid e \rhd e' \in \mathcal{W}\}$ and with output entities $\cup_k O_k \setminus \{e \mid e \rhd e' \in \mathcal{W}\}$.*

What will be the properties of the induced system if we connect two causal systems M_1 and M_2 as illustrated in Figure 4?

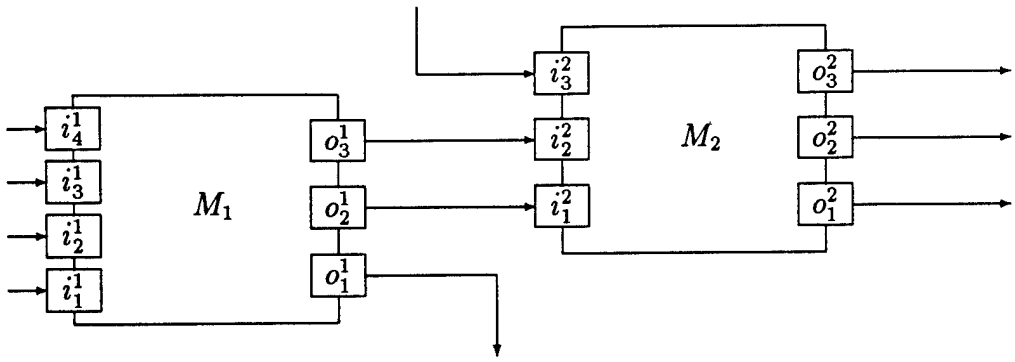


Figure 4 : A simple connection between temporal automata

The following proposition answers this question.

Proposition 3 : *If $M_1 = (I_1, O_1, f_1)$ and $M_2 = (I_2, O_2, f_2)$ are two causal systems and $\mathcal{W}$ is a wiring over M_1, M_2, such that $e \rhd e' \in \mathcal{W}$ implies $e \in O_1$ and $e' \in I_2$ then the system induced by the wiring $\mathcal{W}$ over M_1 and M_2 is a causal system.*

(See appendix A for a proof of this proposition).

What happens in the special case of wiring together temporal automata? Of course, since they are causal systems the result will be a causal system, but will it be a temporal automaton? We have the following proposition.

Proposition 4 : *If $M_1 = (I_1, S_1, O_1, \mathcal{T}_1)$ and $M_2 = (I_2, S_2, O_2, \mathcal{T}_2)$ are two temporal automata and $\mathcal{W}$ is a wiring over M_1, M_2, such that $e \rhd e' \in \mathcal{W}$ implies $e \in O_1$ and $e' \in I_2$ then the system induced by the wiring $\mathcal{W}$ over M_1 and M_2 is a temporal automaton.*

(For a proof of this proposition see appendix A).

Propositions 3 and 4 deal only with unidirectional connections from one causal system to another. We will be more general: complex systems are usually composed of machines with much more complicated interactions, including feedback, as in Figure 5.

For such connections of temporal automata we have the following.

Proposition 5 : *Let M_1,M_2,...,M_n be n temporal automata and $\mathcal{W}$ a wiring over M_1,M_2,...,M_n such that $\forall e$ if $e \rhd e' \in \mathcal{W}$ then e has a durational transduction. The system induced by $\mathcal{W}$ over M_1,M_2,...,M_n is a temporal automaton.*

(See appendix A for a proof of this proposition).

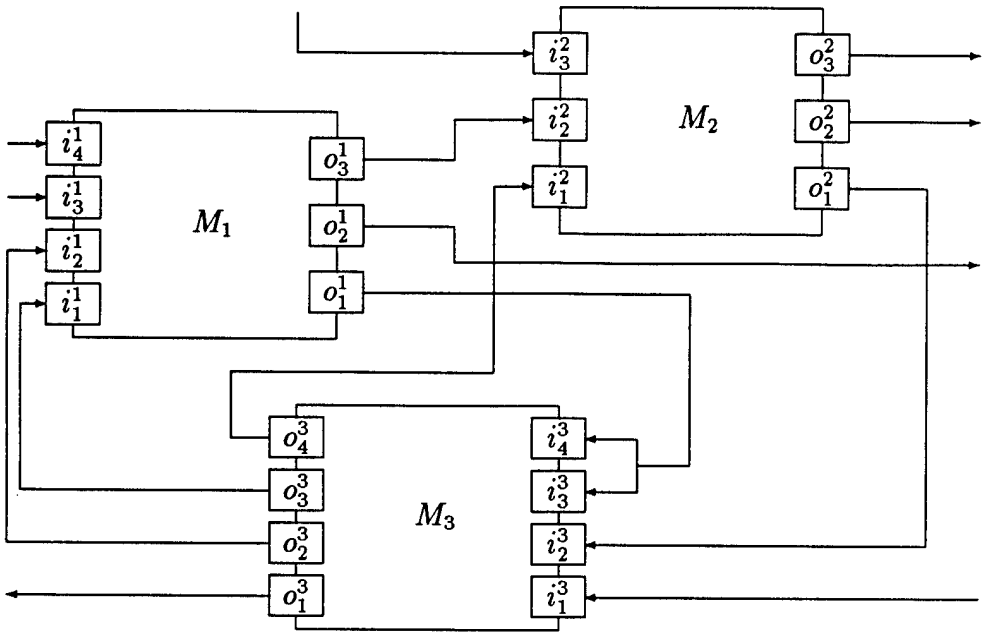


Figure 5 : A more complicated connection between temporal automata

In other words, one can have arbitrarily complex connections and still preserve the property of temporal automaton, as long as there are only durational transductions. Actually this result is a special case of a more general proposition proved in the appendix. That proposition allows nondurational transductions for the connected output as long as they do not introduce loops of zero-delay transductions.

The environment of a machine provides the traces of the unconnected inputs. If we can model the environment itself by a machine, we will obtain a *closed system* where the interaction between a machine and its environment will be represented by the connections of the outputs of the machine to inputs of the environment and the connections of the inputs of the machine from entities of the environment. The formal definition of a closed system is the following.

Definition 13 : *A set of temporal automata M_1,M_2,...,M_n and a set of connections define a* closed system *if for each temporal automaton M_k of this set, all its input entities are connected.*

5 Examples

This section will present two examples of temporal automata.

5.1 A mobile robot

This paragraph contains a small example encoding a simple local navigation procedure in temporal automata. Specifically, we define a temporal automaton M implementing the behavior 'Move forward in the middle of the corridor' for a robot which has two sonars implemented by M_{l_sonar} and M_{r_sonar}.

All the entities used in this example will have a time structure equal to $N_\delta = \{k * \delta \text{ for } k \in N\}$ where $\delta = 50$ milliseconds and the internal and output entities will have a delay δ.

A sonar is here a very simple machine with one input entity *measure* and one output entity *distance*. Their domains are for *measure*, $\{0,1\}$ and for *distance*, R^+. The transduction to *distance* is $(distance, \{measure\}, \delta, (0,0), f)$ with:
$f(measure) =$ if $measure = 0$ then $function\_sonar()$ else 0

The robot M has three inputs d_l, d_r and i. They represent the distance given by the left sonar, the distance given by the right sonar and the order given to the robot. Their domains are $D_{d_l} = D_{d_r} = R^+$ and $D_i = \{move, stop\}$.

The robot has three outputs m,o_l and o_r which are the order to the robot's motor, the order to the left sonar and the order to the right sonar. Their domains are $D_{o_l} = D_{o_r} = \{0,1\}$ and $D_m = \{forward, backward, left, right, stay\}$.

The transduction to o_l is $(o_l, \{i\}, \delta, (0,0), g)$ with:
$g(i) =$ if $i = move$ then 1 else 0.

The transduction to o_r is similar $(o_r, \{i\}, \delta, (0,0), g)$.

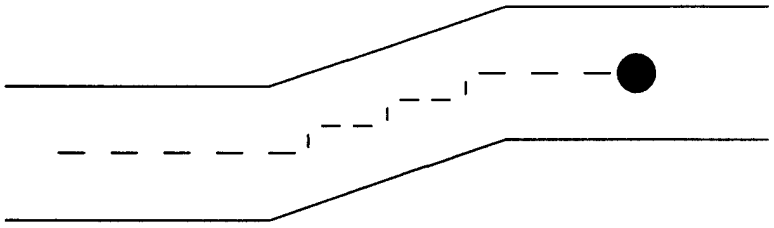
The transduction to m is $(m, \{i, d_l, d_r\}, \delta, (0,0), h)$ with:

$$h(i, d_l, d_r) = \begin{cases} stay & \text{if } i = stop \\ left & \text{if } i = move \text{ and } d_l - d_r > \Delta \\ forward & \text{if } i = move \text{ and } -\Delta \leq d_l - d_r \leq \Delta \\ right & \text{if } i = move \text{ and } d_l - d_r < -\Delta \end{cases}$$

With $\Delta = 20$ centimeters.

The wiring $\mathcal{W} = \{o_r \triangleright measure_{r\_sonar},\ o_l \triangleright measure_{l\_sonar},\ distance_{r\_sonar} \triangleright d_r,\ distance_{l\_sonar} \triangleright d_l\}$ over M, M_{l_sonar} and M_{r_sonar} induces a temporal automaton. This temporal automaton implements the behavior of a robot moving in the middle of a corridor according to the data provided by two sonars.

We can show that for this robot we have $\mid d_l - d_r \mid < \Delta + 4\delta * d' * v$ where d' is an upper bound for the variation of the distance from the wall to the axis of the corridor, and v is the forward speed of the robot. So if for example, $d' = 0.3$ and $v = 1m.s^{-1}$, we have $\mid d_l - d_r \mid <$ 26 centimeters. Then if the corridor is 52 centimeters wider than the robot, the robot is guaranteed collision-free motion.

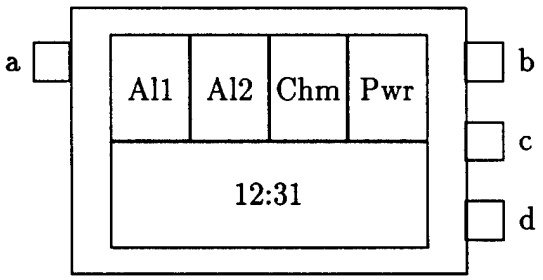


a trajectory of the robot

This temporal automaton is very simple because of the simple environmemt, sensors and effector being modelled. Modelling more realistic hardware and environments would require more entities and more complex transductions, of course.

5.2 A digital watch

This example is borrowed from [5] in which Harel models the behavior of a quartz multi-alarm watch in the Statecharts formalism (see discussion in section 7). We model the same device as a temporal automaton. The watch is depicted in the following figure.



The watch has a main display area and four smaller ones, a beeper, and four control buttons denoted here a, b, c and d. It can display the time (with am/pm or 24 hour time modes) or the date (day of month, month, day of week), has a chime (beeps on the hour if enabled), two independent alarms, a stopwatch and weak battery blinking indication.

The four smaller displays indicate respectively if the alarm1 is enabled, if the alarm2 is enabled, if the chime (beeps every hour) is enabled and if the power is weak.

The main display usually indicates the time. An "event" d makes it displays the date. A loop $time, alarm1, alarm2, chime, stopwatch$ and $time$ is covered when events a occur. If the main display is in a different mode than time for more than 2 minutes it returns automatically to time. In the mode time, if you depress c for more than 2 seconds you switch to the mode update.

The mode update is a loop second, minute, 10 minutes, hour, month, date, day, year, mode which you can cover with c events. In each of the modes a d event increments your time or date by one unit of the mode modulo the next mode. In the update mode an event b returns to the time mode.

If your are in alarm1 or alarm2 modes, an event d change the status of the alarm (enabled, disabled). An event c puts the watch in a mode to update the time of the alarm. The loop is hour, 10 minutes, minute and is covered with event c. An event d in the update mode increments your time of alarm by one unit of the mode modulo the next mode. An event b returns to the mode alarm1 or alarm2.

In the chime mode an event d enabled or disabled the beep every hour.

In the stopwatch mode an event b starts or stops the chronograph. An event d resets it to zero.

The beeper beeps for 30 second if an alarm is enabled and the time is equal to the time of the alarm. During this time the depression of any button stops the beeper. It also beeps for 2 seconds every hour if the chime is enabled.

After this informal description of the watch (slightly different from that in [5]), we will now outline the main principles used for building a temporal automaton with the watch's behavior. The complete definition of the temporal automaton can be found in Appendix B.

There are 5 input entities which are all defined on continous time (a time structure equal to R^+), a, b, c, d and $power$. The first four represent the status of the four buttons and their value will be 1 if the button is depressed and 0 otherwise. The last one is the the information from the battery and can have 3 different values $\{on, weak, dead\}$.

Each of the internal and output entities will be defined on $N_\epsilon = \{k * \epsilon \mid k \in N\}$ where $\epsilon = 10$ milliseconds is the precision of the watch and the minimal time between changes in the watch.

There are 7 output entities. The delay of all the output entities will be 0 since as will be seen they are only transcriptions of internal entities. Four of the output entities will be orders for the display of the four small areas of the watch. There are $al1, al2, chime$ and d_power. The value of the first three will be $\{on, off, none\}$ and for the forth $\{good, weak, none\}$. The fifth output is $light$ which will take its value in $\{on, off\}$, the sixth is $beeper$ with values in $\{beep, silent\}$ and the last is $display$ which will represent what to display on the main area of the watch.

The internal entities can be divided in 4 groups. The first contains entities which keep track of the different modes of the watch. These entities are *main_mode*, *update_mode*, *time_mode*, *chrono_mode*, *updal1_mode* and *updal2_mode*.

The second group contains *alarm1_stat*, *alarm2_stat* and *chime_stat* which encode the status of these features. Their domain will be {*enabled*, *disabled*}.

The third group is used to implement the different time-out behaviors. Such entities will be $2s, 30s$ and $2min$. Their domain will be R.

The last group contains entities which are data. They are *time*, *date*, *al1_time*, *al2_time* and *chrono*. Their domain is R and they will be used by the output entity *display* to draw the main area of the watch.

The entities of the first group encode the different mode of the watch. For example, *main_mode* will take a value in {*tim*, *dat*, *update*, *alarm1*, *alarm2*, *updalarm1*, *updalarm2*, *chime*, *stopwatch*}. If *main_mode* has value *update* it means that we are updating the time or the date of the watch. If it has value *stopwatch*, it means that the stopwatch is being displayed and that the inputs on the buttons are interpreted as orders to the stopwatch. Its transduction is $(main\_mode, \{main\_mode, a, b, c, d, 2s, 2min, update\_mode, updal1\_mode, updal2\_mode\}, \epsilon, (0, tim), f)$. The function f changes the value of *main_mode* for each event a according to the previous value of *main_mode*. It uses the entities $2min$ and $2s$ to implement the return in the time mode after 2 minutes in another mode and the condition for entering the update mode. According to the previous value of *main_mode*, f interpretes the actions on buttons b and c.

The entities of the second group, *alarm1_stat*, *alarm2_stat* and *chime_stat* change value if the entities keeping track of the different modes are in the appropriate mode and the right button is depressed (d actually).

The entities of the third group are similar in principle to the entity implementing a clock presented in section 2.3. They are set to a particular value on some conditions then incremented (or decremented) by ϵ at each cycle. When they reach a particular value they activate a change in other entities.

The entities of the fourth group store data. The entity *time* for example, stores the hour, minute, second, hundredth of second of the current time. In normal mode it is incremented by ϵ every cycle. In the update mode its value is changed according to the events on buttons c and d. The entity *al1_time* stores the time when the beeper will be activated if *alarm1* is enabled. It is set up in the mode *updal1* with buttons c and d.

The output entities are only transcription of internal entities. For example for the four small displays of the watch, the output will be the value of the corresponding mode internal entity if the *power* is not *dead*. The beeper will be activated if the value of *time* matches the value of *al1_time* and *alarm1* is enabled, or *time* matches *al2_time* and *alarm2* is enabled, or *time* is on the hour and *chime* is enabled. For the main display, the output will be the

value of either *time* or *date* or *chrono* or *al1_time* or *al2_time*, depending on the value of *main_mode*.

6 Relation to automata theory

While our model relies on explicit representation of time, classical automata theory hides the notion of time in that of transition. We believe that an explicit expression of time has several advantages when one is concerned with modelling real-time, real-world systems such as robots. This is consistent with the recent trend in the program verification community to incorporate real-time notions into the language for reasoning about the computation (cf. [3]).

At the same time it is important to understand the relation between our model and foundational automata-theoretic models. Our model is obviously an extention of finite automata, but in fact it is a very radical extension. In particular, we have the following property:

Proposition 6 : *Any deterministic one-tape Turing machine (abbreviated DTM) can be simulated by a temporal automaton.*

(See appendix C for a proof of this proposition).

So our model of a temporal automaton has at least the same computational power as a deterministic one-tape Turing machine.

The reverse is not true since our model can represent computation over continuous time, on infinite domains and with an infinite number of entities. However, in most practical cases, we use our framework to model systems where continuous domains or time can be approximated by discrete domains or time. In those cases if the functions involved in the transductions are computable, there exists a deterministic Turing machine able to compute the value of any entity at any time.

7 Relation to some concurrency models

The specific features of our model such as concurrency and composition make it potentially closer in spirit to some of the more recent computational models geared towards concurrent computation. We briefly discuss three of the better known ones, but we aware are of the fact that a large number of other models exist. We are familiar with some of them[3] but, we are sure, not all. One purpose of this document is to solicit comparisons with other

[3]Partly through conversations with David Dill, which we gratefully acknowledge

models which we have not mentioned. The models we do discuss here are Hoare's model of Communicating Sequential Processes, or CSP [7], Milner's Calculus of Communicating Systems, or CCS [10], and Harel's Statecharts [6].

CSP is a mathematical model based on the concept of processes. A process is a set of traces, a trace being a finite sequence of symbols denoting events.[4] A trace represents a possible behavior of a process from its beginning up to some moment in time.

The main characteristics of CSP are:

- One can define the sequential combination and summation of processes.
- You can prefix a process by an event and define transformation on the events alphabet.
- The parallel combination of processes is defined with conditions on the projections of its traces on the processes events alphabet.
- The communication between two processes is synchronous.
- The data structures are implemented via processes.

The main concept of CCS is an algebra of behaviors. These behaviors are built from a set of names representing the possible events and the operations defined on this algebra.

Briefly, the major features of CCS are:

- Each name of an event is associated with an input event and an output event. The communication between behavior is synchronous.
- One can define the relabelling, the restriction of a behavior and add a prefix to a behavior. One can define the summation and the parallel composition of behaviors.
- Behaviors are defined usually by equations, often recursive, which may contain programming constructs such as conditionals.
- The concept of a communication tree is defined as an interpretation of behaviors, and what it means for a set of behaviors to be observation equivalent is defined.

We have not carried out a formal comparison between temporal automata and either CSP or CCS, but several important differences stand out. Chief among these differences is the treatment of time. Both CSP and CCS rely on the basic notion of 'sequence'; the only temporal information representable is precedence. Furthermore, in both models all communication is synchronous, unlike in our model. On the other hand we share with both

[4]The general sense of 'trace' is the same as in our model, but the specifics are different. In particular, in CSP (and CCS) there is no temporal information in a trace other than the order among events.

models the property that any communication of information requires that the participants in the communication be named.

CSP and CCS are also different from temporal automata in that they are more "observation based" in spirit; the emphasis is on describing the various stages of the processes, not the causal mechanism generating the process.

In this sense temporal automata are closer to Statecharts. Statecharts are a visual formalism for the behavioral description of complex systems. They extend classical state-diagrams in several ways, while retaining their formality and visual nature. Like finite state machines, Statecharts are based on states, events and conditions, with combinations of the latter two causing transitions between the former. Both states and transitions can be associated in various ways with outputs events, called actions, which can be triggered either by executing a transition or by entering, exiting or simply being in a state. The system's inputs are thus the external events and its outputs are the external actions.

The main features of Statecharts are:

1. They allow hierarchical description of the states. One can use any combination of *xor* and *and* between states as a new state.
2. One can specify default entrance or historic entrance into a *xor* state.
3. One can specify delay or timeout to exit from a state.
4. One can encode actions and conditional entrances.

The internal states, the wiring and the hierarchical description of Statecharts are very similar to those in temporal automata. In fact, there is a straightforward translation of any Statechart into a temporal automaton. Furthermore, there exists a graphic interface to describe complex systems with Statecharts which is similar in spirit to the simulator for temporal automata described in section 9.

The main difference between Statecharts and temporal automata again has to do with the representation of time; whereas we allow arbitrary temporal structures and asynchronous communication, Statecharts assume a discrete and synchronous structure of time.

8 Relation to some recent models in AI

Several researchers in AI have recently employed models of concurrent computation, driven by an interest in representing physical agents embedded in a physical environment. Indeed, this has also been our motivation, and temporal automata is intended as the target language

for the compiler in the AOP project [17]. Unlike in our model, most AI researchers have adopted a standard circuit-like framework.

One example is Rosenschein and Kaelbling's *situated automata* [14]. The main purpose of situated automata is to design systems with provable epistemic properties and with provable reaction times. The applications of this formalism are the conception of highly reactive artificial-intelligence systems such as intelligent robots. The component of situated automata relevant to the present comparison is the low-level machine description, which is done in terms of switching circuits. The elements of the circuit are either functional elements which compute instantaneously a function of their inputs, or delay elements which delay their input by a constant time. Thus in that framework, the model of time is that of the integers, and all connections are synchronous.

Similar properties are true of Nilsson's *action networks* [12]. ACTNET is a language for computing goal-achieving actions that depends dynamically on sensory and stored data. ACTNET programs can be used to control the actions of a robot in a dynamic environment.

The ACTNET language is based on the concept of action networks [11]. An action network is a forest of logical gates that select actions in response to sensory and stored data. The elementary unit of an action net implements a logical *and* gate as illustrated in figure 6.

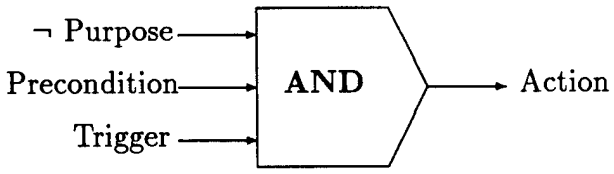


figure 6 : action unit

In contrast to our framework, ACTNET allows only logical *and* entities, all units are synchronous, and all act instantaneously and simultaneously (i.e. their output is updated as soon as their inputs change). In a way, ACTNET is a synchronous language similar to Esterel [1].

ACTNET does not represent temporal behavior, nor does it permit loops among action units. Thus to implement feedback loops one must rely on the environment which updates the sensory and stored data; ACTNET itself does not control those data.

In [4] Genesereth discusses a continuum of *agents*. His main interest lies in high-level, informable agents capable of acting on partial plans. In our context, however, most relevant are the lowest-level agents, which are each a finite automaton, and which interact by connecting the input of one agent to the output of another. Exactly as in our model, Genesereth accords equal status to the agent and the environment, treating the environment as simply another agent. Although he does not pursue it, it is possible in his framework to aggregate

agents and create a hierarchical structure. The temporal dimension, however, is absent in the finite-automata representation.

As a final example from AI, we mention Brooks' creatures [2]. Brooks has explored the thesis that natural intelligence evolved in a bottom-up fashion, and that artificial intelligence should be created in the same way. For this purpose he has developed a so-called *subsumption architecture*, in which simple behaviors are encoded and then used to create more complicated ones. All behaviors are implemented in a structure that is essentially a switching circuit. In particular, no timing information is present and all communication is synchronous.

9 A programming language based on temporal automata

Our framework can be presented as a language for the description of machines. There are two advantages to presenting our model as a programming language. First of all, we can use this language to describe temporal automata and have them interpreted and simulated on a computer. We can also use this language as a target language for the translation of high level specifications of an agent. By this process we tranform a high level specification into a reactive system composed of temporal automata.

To study and try to achieve an automatic translation we need a formal definition of this language for the descriptions of temporal automata.

This language is defined in a BNF by the following grammar. In the BNF notations used here, { E }$^+$ means one or more occurrences of E; { E }* means zero or more occurrences of E, and [E] indicates that E is optional. Reserved words are written in boldface.

<program>	::=	{ <declaration> }* { <def machine> }$^+$ { <def connection> }* { <def input> }*
<def machine>	::=	**machine** <ident> **input** { <def entity> }* **state** { <def entity> <dependence>}* **output** { <def entity> <dependence>}*
<def connection>	::=	**connect** (<ident> <ident>) (<ident> <ident>)
<def entity>	::=	**entity** <ident> <type> <t structure>
<declaration>	::=	<decl type>
<decl type>	::=	**type** <ident> <type>

<type>	::=	<type predefined> |<type name> |**list**(<type>) |({ <type> }$^+$)
<type name>	::=	<ident>
<t structure>	::=	**timeN** | **timeR** | **timeD** <real> | **time** <code-function>
<dependence>	::=	**depend** <ident> ({ <ident> }$^+$) <functionT> <function>
<functionT>	::=	<code-function> |((0 <object>) {(<real> <object>)}*)
<function>	::=	({ <ident> }*) <code-function>
<code-function>	::=	a function written in Lisp
<object>	::=	<integer> | <real> | <boolean> | <character> | ({<object> }*)
<integer>	::=	[-] <digit> {<digit>}*
<real>	::=	[-] <digit> {<digit>}*.<digit> {<digit>}*
<boolean>	::=	**true** | **nil**
<character>	::=	any printable character
<digit>	::=	0 | ... | 9
<ident>	::=	<letter> {<letter> | <digit>}*
<letter>	::=	a | ... | z | A | ... | Z

The semantics associated with this grammar in terms of temporal automata is rather clear from the language. For example, a <function> is the definition of the function f of a transduction. The key word **connect** introduces 4 identifiers which are the names of the machine and the entity from which the connection starts and the names of the machine and the entity which are been connected. (The complete semantics are not given in the report).

A program which simulates the temporal evolution of temporal automata has been written. This simulator takes a description of temporal automata written in the language described above. Then it simulates the evolution of the machine given a trace for inputs from the environment. The environment inputs can be either given interactively by the user or

defined by a function.

Of course, time in the simulator is discrete but the user can use any precision for time he or she needs. Except for this restriction, that domains and time structures are discrete, all the features of the model are implemented in the simulator.

The interface of the simulator allows the user to define temporal automata interactively instead of describing them in the language. When a machine is defined the user can simulate the machine several times for different traces of the inputs from the environment.

The simulator is written in Lisp and the user has access to this language to define customizations or special interfaces for a particular simulation.

For more details on this simulator see [9].

10 Summary and concluding remarks

In this report, we have introduced a framework for describing machines and related it to other work in the area. We believe this to be a powerful formalism in which to represent the low-level specifications of agents. The main advantages of this model are:

- Time is explicit. This feature allows us to represent asynchronous machines as well as machines with continuous or discrete time. We can represent all temporal phenomena directly in our framework.
- Each entity has a delay. This provides a realistic way in which to represent computation by hardware. As each entity has a delay, we can represent by an entity both very fast and simple computations and very long and complex ones.
- Modularity. One can define several machines, connect them and consider again the whole system as a machine. This capability to define smaller systems and then assemble them together is very useful when you want to represent large complex systems.
- Several levels of abstraction. Our framework is convenient for representing different levels of abstraction of a system. One can express by an entity either very basic operations like boolean functions or complex one. One can therefore treat a machine as a black box, and then selectively refine the model.

We note that our computational model deals only with deterministic systems whereas most existing models include also nondeterministic behaviors. This is not accidental, as we aim in our model to capture the full causal mechanism, whereas nondeterminism reflects ignorance of some aspects of the machine. However, although we have so far had no reason for doing so, it is easy to define either nondeterministic or stochastic versions of our model, by modifying the definition of a transduction.

Further theoretical investigations of temporal automata are possible, including comparisons with other models and investigation of complexity issues. Our own interest lies in testing the practical usefulness of the model. In particular, we intend to use it as a target into which to compile cognitive agent programs within the Agent-Oriented Programming framework.

Acknowledgements

We are grateful to David Dill and Fangzhen Lin for valuable discussions.

References

[1] G. Berry, P. Gonthier, G Gonthier. *"Synchronous programming of reactive systems: an introduction to Esterel."*.
Rapport de recherche, INRIA (France) 1987.

[2] Rodney Brooks. *"Intelligence without representation"*
Workshop on Foundations of Artificial Intelligence, June 1987.

[3] R. Alur, D. Dill. *"Automata for modeling real-time systems"*
Proceedings of ICALP, 1990.

[4] Michael Genesereth. *"Agents"*
Working paper Logic-89-6, Computer Science Department, Stanford University.

[5] David Harel. *"Statecharts: a visual approach to complex systems"*
Technical report, Weizmann Institute of Science, CS86-02.

[6] David Harel, D. Drusinsky. *"Using statecharts for hardware description and synthesis"*.
IEEE transactions on computer-aided design vol. 8 no. 7, July 1989.

[7] C.A.R. Hoare. *"A model for comminucating sequential processes"*.
Technical report, Oxford University Computing Laboratory.

[8] Leslie Kaelbling. *"Goals as parallel program specifications"*
Proceedings of AAAI, 1988.

[9] Jean-François Lavignon. *"A simulator for temporal automata"*
Programer's manual, Robotics Laboratory, Stanford University.

[10] Robin Milner. *"A calculus of communicating systems"*
Lecture Notes in Computer Science 92, Springer-Verlag.

[11] Nils Nilsson. *"Action networks"*. Proceedings from the Rochester planning workshop: From formal system to practical systems. Tenenberg,J et al., Universty of Rochester, New-York 1989.

[12] Nils Nilsson, Rebecca Moore, Mark Torrance. *"ACTNET: an action-network language and its interpreter"*.
Preliminary report, Stanford University 1990.

[13] François Reuleaux. *"Cinematique paire"*

[14] Stanley Rosenschein, Leslie Kaelbling. *"The synthesis of machines with provable epistemic properties"*
Proceeding of the 1986 Conference on Theoretical aspects of reasoning about knowledge, Morgan Kaufmann Publishers, 1986.

[15] Stanley Rosenschein *"Synthesizing information-tracking automata from environment descriptions"*
Paper, 1990.

[16] M.W. Shields *" An introduction to Automata Theory"*
Blackwell scientific publications 1987

[17] Yoav Shoham *"Agent-oriented programming"*
Technical report, Computer Science Department, Stanford University.

Appendix A

Proposition 1 : *A transduction* $(e, I, \delta, f_{init}, f)$ *where either* $e \notin I$ *or* $\delta \neq 0$ *defines a causal function from* $\Pi_{i \in I \setminus \{e\}} V_i$ *to* V_e

Proof : Let us first treat the trivial case where $e \notin I$.
$\forall (v_{i_1}, v_{i_2}, ...), (v'_{i_1}, v'_{i_2}, ...) \in \Pi_{i \in I} V_i$, $\forall t \in T_e$
either $t < \delta$ and then $v_e(t) = f_{init}(t) = v'_e(t)$
or $t \geq \delta$ and then
if $\forall i_k \in I$, $v_{i_k} = v'_{i_k}$ on $T_{i_k} \cap [0, t]$ then $\forall i_k \in I$, $\overline{v_{i_k}} = \overline{v'_{i_k}}$ on $[0, t]$
so $v_e(t) = f(\overline{v_{i_1}}(t-\delta), \overline{v_{i_2}}(t-\delta), ...)$ is equal to
$v'_e(t) = f(\overline{v'_{i_1}}(t-\delta), \overline{v'_{i_2}}(t-\delta), ...)$

Let us suppose now that $e \in I$ and $\delta > 0$

Let us show that the transduction defines a function h from $\Pi_{i \in I \setminus \{e\}} V_i$ to V_e. We suppose that $e = i_1$ and denote by $\tilde{v}$ an element of $\Pi_{i \in I \setminus \{e\}} V_i$ and by $\overline{\tilde{v}}$ the element of $\Pi_{i \in I \setminus \{e\}} R^+ \rightarrow D_i$ associated with $\tilde{v}$.

We define by induction on k a family of functions h_k from $\Pi_{i \in I \setminus \{e\}} V_i$ to $T_e \cup [0, k*\delta) \rightarrow D_e$ and prove by induction on k that
P_k $[\forall\, \tilde{v}\ \tilde{v}'\ \forall t < k * \delta\ [\tilde{v} = \tilde{v}' \text{ on } [0, t]] \Rightarrow h_k(\tilde{v})(t) = h_k(\tilde{v}')(t)]$

For $k = 1$ we define h_1 by:
$\forall \tilde{v} \in \Pi_{i \in I \setminus \{e\}} V_i$ $\forall t \in T_e \cup [0, \delta)$, $h(\tilde{v})(t) = f_{init}(t)$. Clearly P_1 holds.

Suppose we have h_k function $\Pi_{i \in I \setminus \{e\}} V_i$ to $T_e \cup [0, k * \delta) \rightarrow D_e$ which verifies P_k. We define h_{k+1} by:
if $t \in T_e \cup [0, k * \delta)$ $h_{k+1}(\tilde{v})(t) = h_k(\tilde{v})(t)$.
if $t \in T_e \cup [k * \delta, (k+1) * \delta)$ $h_{k+1}(\tilde{v})(t) = f(\overline{h_k(\tilde{v})}(t-\delta), \overline{\tilde{v}}(t-\delta))$.
Clearly P_k implies P_{k+1}

The function h from $\Pi_{i \in I \setminus \{e\}} V_i$ to V_e is defined by:
$\forall t \in T_e$, $h(\tilde{v})(t) = h_k(\tilde{v})(t)$ where $k \mid t \in [(k-1) * \delta, k * \delta)$
The causality of h follows from the properties P_k. □

Proposition 2 : *If* M *is a temporal automaton* $(I, S, O, \mathcal{T})$ *then there exists a unique causal function from* $\Pi_{i \in I} V_i$ *to* $\Pi_{e \in S \cup O} V_e$ *which satisfies the transductions equations.*

Proof : We must show that there is a function $\mathcal{F}$ from $\Pi_{i \in I} V_i$ to $\Pi_{e \in S \cup O} V_e$ which satisfies the equations of the transductions. This means that if we have the transduction $(e, A, \delta, f_{init}, f)$ we want:
$\forall \tilde{v} \in \Pi_{i \in I} V_i$ $\forall t \in [0, \delta) \cap T_e$ $\mathcal{F}_e(\tilde{v})(t) = f_{init}(t)$
$\forall t \in T_e$ $t \geq \delta$ $\mathcal{F}_e(\tilde{v})(t) = f(\overline{\mathcal{F}_{a_1}(\tilde{v})}(t-\delta), \overline{\mathcal{F}_{a_2}(\tilde{v})}(t-\delta), ...)$ where $\mathcal{F}_e$ represents the projection of $\mathcal{F}$ on V_e.

First we need to introduce a function *ord* from the set of entities $I \cup S \cup O$ to N. This function will have the property that $ord(e) = sup\{ord(e'), e' \in \mathcal{D}^0(e)\} + 1$. It is possible to define such a function because $\mathcal{D}^0$ does not have any cycles.

Let us show how. We define the family of functions d_n by:
$d_0(e) = 0$ if $\mathcal{D}^0(e) = \emptyset$; $d_0(e) = \omega$ otherwise.
$d_{n+1}(e) = d_n(e)$ if $d_n(e) \neq \omega$
$d_{n+1}(e) = n + 1$ if $\forall e' \in \mathcal{D}^0(e)\ d_n(e') \neq \omega$ and $d_n(e) = \omega$
$d_{n+1}(e) = \omega$ otherwise.
We show that $\forall e\ \exists n \mid d_n(e) \neq \omega$. Let us suppose the contrary. Then for one entity e we have $\forall n \in N d_n(e) = \omega$. Then $\forall n \exists a(n) \mid a(n) \in \mathcal{D}^0(e)$ and $d_{n-1}(a(n)) = \omega$. But $\mathcal{D}^0(e)$ is finite so $\exists a_1 \in \mathcal{D}^0(e) \mid \forall n \exists k > n \mid d_k(a_1) = \omega$. So according to the definition of the family d_n, we have $\forall n\ d_n(a_1) = \omega$. If we iterate the same demonstration for a_1 than for $e = a_0$, we can construct a sequence $(a_0, a_1, a_2, ..) \mid \forall i\ a_{i+1} \in \mathcal{D}^0(a_i)$. So $\forall i\ \{a_{i+1}, a_{i+2}, ...\} \subset \Delta^0(a_i)$. But $\Delta^0(e)$ is finite (included in $\Delta(e)$). So there must be $a_i = a_j, i < j$. But then $a_i = a_j \in \Delta^0(a_i)$ which is impossible. So we have demonstrated that $\forall e \exists n_e \mid d_{n_e}(e) \neq \omega$. We define $ord(e) = d_{n_e}(e)$. So if $ord(e) = k > 0$ then by the definition of d_n we have $ord(e) = sup\{ord(e'), e' \in \mathcal{D}^0(e)\} + 1$.

Now we will prove that $\forall e \in S \cup O$ there is a causal function from $\Pi_{i \in I} V_i$ to $\Pi_{a \in \Delta(e) \cup \{e\} \setminus I} V_a$ which satisfies the equations of the transductions. We know that $(\Delta(e) \cup \{e\}) \setminus I = A$ is finite because M is a temporal automaton. So $\exists \delta_A > 0 \mid \forall a \in A \mid \delta_a \neq 0$ we have $\delta_A < \delta_a$.

We will prove by induction on k that:
$\forall k > 1\ \exists f_k^A$ a causal function from $\Pi_{i \in I} V_i$ to $\Pi_{a \in A}(T_a \cap [0, k * \delta_A) \rightarrow D_a)$ which satisfies the equations of the transductions.

In the case $k = 1$:
if $\mathcal{D}^0(a) = \emptyset$ i.e. $ord(a) = 0$ we define $f_{1_a}^A$ by $\forall \tilde{v} \in \Pi_{i \in I} V_i\ \forall t \in [0, \delta_A) \cap T_a\ f_{1_a}^A(\tilde{v})(t) = f_{init_a}(t)$. $f_{1_a}^A$ is causal because of this definition.
if $ord(a) = 1$ $\forall b \in \mathcal{D}^0(a)$ either $b \in I$ or we have just defined $f_{1_b}^A$. So we can define $\forall t \in [0, \delta_A) \cap T_a\ f_{1_a}^A(\tilde{v})(t) = f_a(\overline{r_{b_1}(\tilde{v})}(t), \overline{r_{b_2}(\tilde{v})}(t), ..)$. Where $r_b(\tilde{v})(t) = v_b(t)$ if $b \in I$ and $r_b(\tilde{v})(t) = f_{1_b}^A(\tilde{v})(t)$ if $b \in A$. By this definition $f_{1_a}^A$ is causal and satisfies the equations of the transductions.
if $ord(a) = 2$ we can iterate the same reasoning and define $f_{1_a}^A$.

As A is finite $\exists K \mid \forall a \in A\ K > ord(a)$. So after a finite number of iterations we define a causal function f_1^A from $\Pi_{i \in I} V_i$ to $\Pi_{a \in A}(T_a \cap [0, \delta_A) \rightarrow D_a)$ which satisfies the equations of the transductions.

Now suppose that this is true for k. We define f_{k+1}^A in the following way:
if $ord(a) = 0$, $\forall b \in \mathcal{D}(a)$ we denote by $r_b(\tilde{v})$, v_b if $b \in I$ or $f_{k_b}^A(\tilde{v})$ if $b \in A$. $\forall \tilde{v} \in \Pi_{i \in I} V_i$
if $t \in [0, k * \delta_A) \cap T_a\ f_{k+1_a}^A(\tilde{v})(t) = f_{k_a}^A(\tilde{v})(t)$
if $t \in [k * \delta_A, (k+1) * \delta_A) \cap T_a \cap [0, \delta_a)\ f_{k+1_a}^A(\tilde{v})(t) = f_{init_a}(t)$
if $t \in [k * \delta_A, (k+1) * \delta_A) \cap T_a \cap [\delta_a, \infty)\ f_{k+1_a}^A(\tilde{v})(t) = f_a(\overline{r_{b_1}(\tilde{v})}(t - \delta_a), \overline{r_{b_2}(\tilde{v})}(t - \delta_a), ...)$.

As $\delta_a > \delta_A$, $t - \delta_a < k * \delta_A$. This definition is causal and satisfies the transductions equations since all its components do.

If $ord(a) = 1$, then $\forall b \in \mathcal{D}^0(a)$ either $b \in I$ and we denote $r_b(\tilde{v}) = v_b$ or $ord(b) = 0$ and we denote by r_b the function $f^A_{k+1_b}$ that we have just defined.
if $t \in [0, k * \delta_A) \cap T_a$ $f^A_{k+1_a}(\tilde{v})(t) = f^A_{k_a}(\tilde{v})(t)$
if $t \in [k * \delta_A, (k+1) * \delta_A) \cap T_a$ $f^A_{k+1_a}(\tilde{v})(t) = f_a(\overline{r_{b_1}(\tilde{v})}(t), \overline{r_{b_2}(\tilde{v})}(t), ..)$.
This definition implies that $f^A_{k+1_a}$ is causal and satisfies the transductions equations.
As in the definition of f^A_1 we can iterate this construction on the value of ord. We will stop because A is finite.
So we can define the function f^A_{k+1}.

So by induction we have $\forall k > 1$ $\exists f^A_k$ a causal function from $\Pi_{i \in I} V_i$ to $\Pi_{a \in A}(T_a \cap [0, k * \delta_A) \rightarrow D_a)$ which satisfies the equations of the transductions.

We will now construct f^A a causal function from $\Pi_{i \in I} V_i$ to $\Pi_{a \in A} V_a$ which satisfies the equations of the transductions. We define f^A by:
$\forall \tilde{v} \in \Pi_{i \in I} V_i$ $\forall t \in [(k-1) * \delta_A, k * \delta_A)$
$f^A_e(\tilde{v})(t) = f^A_{k_e}(\tilde{v})(t)$

This definition implies that f^A is causal and that it satisfies the transductions equations because of the construction of the f^A_k.

Let us show now that if $e \in (\Delta(a) \cup \{a\} \setminus I) \cap (\Delta(b) \cup \{b\} \setminus I)$ then we have $f^A_e = f^B_e$, where $A = \Delta(a) \cup \{a\} \setminus I$ and $B = \Delta(b) \cup \{b\} \setminus I$. Let us suppose the contrary. Then $\exists \tilde{v_0} \in \Pi_{i \in I} V_i$ $\exists t_0 \mid f^A_e(\tilde{v_0})(t_0) \neq f^B_e(\tilde{v_0})(t_0)$.

But we have a transduction $(e, \mathcal{D}(e), \delta_e, f_{init_e}, f_e)$ and f^A and f^B satify the equation of that transduction.
if $\delta_e = 0$ we have $f_e(\overline{f^A_{e'}(\tilde{v_0})}(t_0), ...) \neq f_e(\overline{f^B_{e'}(\tilde{v_0})}(t_0), ...)$. So $\exists e' \in \mathcal{D}^0(e)$ $\exists t' \leq t_0 \mid f^A_{e'}(\tilde{v_0})(t') \neq f^B_{e'}(\tilde{v_0})(t')$ and $ord(e') < ord(e)$.
if $\delta_e > 0$ we have $f_e(\overline{f^A_{e'}(\tilde{v_0})}(t_0 - \delta_e), ...) \neq f_e(\overline{f^B_{e'}(\tilde{v_0})}(t_0 - \delta_e), ...)$. So $\exists e' \in \mathcal{D}(e)$ $\exists t' \leq t_0 - \delta_e \mid f^A_{e'}(\tilde{v_0})(t') \neq f^B_{e'}(\tilde{v_0})(t')$.
Note that $t_0 < \delta_e$ is impossible because f^A_e and f^B_e are both then equal to f_{init_e}.

If we have $\delta_0 = min(\delta_A, \delta_B)$ then we have show that:
if $\exists e \in A \cap B$ such that
$\exists \tilde{v_0} \in \Pi_{i \in I} V_i$ $\exists t_0 \mid f^A_e(\tilde{v_0})(t_0) \neq f^B_e(\tilde{v_0})(t_0)$
then $\exists e' \in A \cap B$ $\exists t'_0$ such that either $ord(e') < ord(e)$ or $t'_0 \leq t_0 - \delta_0$ and $f^A_{e'}(\tilde{v_0})(t'_0) \neq f^B_{e'}(\tilde{v_0})(t'_0)$

We can iterate and construct an infinite sequence. But as ord has an upper bound on $A \cup B$ we must decrease t_0 by at least δ_0 an infinite number of time. So the time should be negative which is impossible. So we have $f^A_e = f^B_e$.

Now we can construct $\mathcal{F}$ by defining $\mathcal{F}_e = f^{\Delta(e) \cup \{e\}}_e$. Then $\mathcal{F}$ is a causal function which satisfies the equations of the transductions since the functions f^A do.

We can prove that $\mathcal{F}$ is unique with the same arguments used to prove that $f_e^A = f_e^B$.□

Proposition 3 : *If $M_1 = (I_1, O_1, f_1)$ and $M_2 = (I_2, O_2, f_2)$ are two causal systems and $\mathcal{W}$ is a wiring over M_1, M_2, such that $e \triangleright e' \in \mathcal{W}$ implies $e \in O_1$ and $e' \in I_2$ then the system induced by the wiring $\mathcal{W}$ over M_1 and M_2 is a causal system.*

Proof : Let O be the subset of output entities of M_1 that are connected to M_2.

Let I be the subset of input entities of M_2 that are connected from M_1.

We will prove that $(I_1 \cup (I_2 \setminus I), S_1 \cup S_2 \cup I, (O_1 \setminus O) \cup O_2)$ and a causal function defined in terms of f_1,f_2, and the connections is a causal system.

S_1 is related $I_1 \cup (I_2 \setminus I)$ by a causal function since it is to I_1 by f_1.

$S_2 \cup I$ is related to $I_2 = I \cup (I_2 \setminus I)$ by a causal function f_2, but I is related to O by the connections functions and O is related to I_1 by f_1. So, $S_2 \cup I$ is related to $I_1 \cup (I_2 \setminus I)$ by a causal function.

$O_1 \setminus O$ is related to I_1 by f_1.

O_2 is related to $I_2 = I \cup (I_2 \setminus I)$ by f_2 so with the same argument as for S_2, O_2 is related to $I_1 \cup (I_2 \setminus I)$ by a causal function. □

Proposition 4 : *If $M_1 = (I_1, S_1, O_1, \mathcal{T}_1)$ and $M_2 = (I_2, S_2, O_2, \mathcal{T}_2)$ are two temporal automata and $\mathcal{W}$ is a wiring over M_1, M_2, such that $e \triangleright e' \in \mathcal{W}$ implies $e \in O_1$ and $e' \in I_2$ then the system induced by the wiring $\mathcal{W}$ over M_1 and M_2 is a temporal automaton.*

Proof : Let O be the subset of output entities of M_1 that are connected to M_2.

Let I be the subset of input entities of M_2 that are connected from M_1.

We will prove that $(I_1 \cup (I_2 \setminus I), S_1 \cup S_2 \cup I \cup O, (O_1 \setminus O) \cup O_2)$ with appropriate transductions is a temporal automaton equivalent to the system induced by the wiring of M_1 and $M2$.

For $e \in E = S_1 \cup S_2 \cup I \cup O \cup (O_1 \setminus O) \cup O_2$ we take the following transduction:
- if $e \in S_1 \cup S_2 \cup O_1 \cup O_2$ the same transduction as in $M1$ or $M2$,
- if $e \in I$ then $\exists o \in O \mid o \triangleright e$ and we take the transduction $(e, \{o\}, 0, 0, id)$ where id is the identity function.

It is obvious that $\forall e \in E$, $\Delta(e)$ is finite.

Let us now show that $\forall e \in E, e \notin \Delta^0(e)$.
If $e \in S_1 \cup O_1$ this follows from the fact that M_1 is a temporal automaton.
If $e \in I$ then $\mathcal{D}^0(e) = \{o\}$ and so $\Delta^0(e) = \{o\} \cup \Delta^0(o)$ which is included in $I_1 \cup S_1 \cup \{o\}$ so $e \notin \Delta^0(e)$.
If $e \in S_2 \cup O_2$ then $\Delta^0(e) = \Delta^0_{M_2}(e) \cup \cup_{i \in I \cap \Delta^0_{M_2}(e)} \Delta^0(i)$. But $e \notin \Delta^0_{M_2}(e)$ because M_2 is a temporal automaton and $\cup_{i \in I \cap \Delta^0_{M_2}(e)} \Delta^0(i) \subset (I_1 \cup S_1 \cup O)$, so $e \notin \Delta^0(e)$.

The system we have defined with our new transductions is then a temporal automaton. Furthermore it is equivalent to the system induced by the wiring of M_1 and M_2. Each entity of $S_1 \cup S_2 \cup O_1 \cup O_2$ has the same transduction and if i is an entity of I its transduction implies that $\forall t \in T_i\ v_i(t) = \overline{v_o}(t)$ which is exactly the behavior induced by the connection. □

Proposition 5 : *Let M_1,M_2,...,M_n be n temporal automata and $\mathcal{W}$ a wiring over M_1,M_2,...,M_n such that $\forall e$ if $e \rhd e' \in \mathcal{W}$ then e has a durational transduction. The system induced by $\mathcal{W}$ over M_1,M_2,...,M_n is a temporal automaton.*

Proof : We denote by $\bar{O}_i$ the set of output entities of M_i which are connected to other entities and by $\bar{I}_i$ the set of input entities of M_i which are connected from other entities.

We will prove that $\bigcup_{i=1}^{n}(S_i \cup \bar{O}_i \cup \bar{I}_i) \cup \bigcup_{i=1}^{n}(O_i \setminus \bar{O}_i))$ is related to $\bigcup_{i=1}^{n}(I_i \setminus \bar{I}_i)$ by a causal function.

We will show that the system $(\bigcup_{i=1}^{n}(I_i \setminus \bar{I}_i), \bigcup_{i=1}^{n}(S_i \cup \bar{O}_i \cup \bar{I}_i), \bigcup_{i=1}^{n}(O_i \setminus \bar{O}_i))$ and the sets of transductions of M_i and the transductions $\forall e \in \bigcup_{i=1}^{n}\bar{I}_i\ \ (e, \{e'\}, 0, 0, id)$, where e' is such that $e' \in \bigcup_{i=1}^{n}\bar{O}_i$ and $e' \rhd e$ is a temporal automaton.

For this system if $\forall e \in \bigcup_{i=1}^{n}(S_i \cup O_i \cup \bar{I}_i)$ we have $\Delta(e)$ finite and $e \notin \Delta^0(e)$ then by definition the system is a temporal automaton.

Let us show that the case where $\forall o_i \in \bar{O}_i$, o_i has a durational transduction is a particular case where this property holds.

As the number of connection is finite, $\forall e \in \bigcup_{i=1}^{n}(S_i \cup O_i \cup \bar{I}_i)$ the cardinality of $\Delta(e)$ is at most increased by (number of connections)$*max_{e \in \cup \bar{O}_i}$(cardinal($\Delta_{old}(e)$)).

Now $\forall e \in \bigcup_{i=1}^{n}\bar{I}_i \mid e' \rhd e$ we have $\Delta^0(e) = \{e'\}$ because $\Delta^0(e') = \emptyset$ since e' has a durational transduction.

Then $\forall e \in \bigcup_{i=1}^{n}(S_i \cup O_i)$, if $e \in S_i \cup O_i$, $\Delta^0(e)$ is equal to the union of $\Delta^0_{M_i}(e)$ and eventually a subset of $\{a \in \bar{I}_i, a' \in \cup_j \bar{O}_j \mid a' \rhd a\}$. So $e \notin \Delta^0(e)$.

We have proof than the system $\bigcup_{i=1}^{n}(I_i \setminus \bar{I}_i), \bigcup_{i=1}^{n}(S_i \cup \bar{O}_i \cup \bar{I}_i), \bigcup_{i=1}^{n}(O_i \setminus \bar{O}_i))$ and the sets of transductions of M_i and the transductions $\forall e \in \bigcup_{i=1}^{n}\bar{I}_i \ (e, \{e'\}, 0, 0, id)$ (where e' is such that $e' \in \bigcup_{i=1}^{n}\bar{O}_i$ and $e' \triangleright e$) verifies the definition of a temporal automaton.□

Appendix B

In this appendix we present the temporal automaton implementing the digital watch described in section 5.2.

There are 5 input entities which are all defined on continous time (time structure equal to R^+), a, b, c, d and *power*. The first four represent the status of the four buttons and their value will be 1 if the button is depressed and 0 otherwise. The last one is the the information from the battery and can have 3 different values $\{on, weak, dead\}$.

Each of the internal and output entities will be defined on $N_\epsilon = \{k * \epsilon \mid k \in N\}$ where $\epsilon = 10$ milliseconds is the precision of the watch and the minimal time between changes in the watch.

There are 7 output entities. The delay of all the output entities will be 0 since as you will see they are only transcriptions of internal entities. Four of the output entities will be orders for the display of the four small areas of the watch. There are $al1, al2, chime$ and d_power. The value of the first three will be $\{on, off, none\}$ and for the forth $\{good, weak, none\}$. The fifth output is *light* which will take its value in $\{on, off\}$, the sixth is *beeper* with values in $\{beep, silent\}$ and the last is *display* which will represent what to display on the main area of the watch.

The internal entities can be divided in 4 groups. The first contains entities which keep track of the different modes of the watch. These entities are *main_mode*, *update_mode*, *time_mode*, *chrono_mode*, *updal1_mode* and *updal2_mode*.

The second group contains *alarm1_stat*, *alarm2_stat* and *chime_stat* which encode the status of these features. Their domain will be {*enabled*, *disabled*}.

The third group is used to implement the different time-out behaviors. Such entities will be $2s, 30s$ and $2min$. Their domain will be R.

The last group contains entities which are data. They are *time*, *date*, *al1_time*, *al2_time* and *chrono*. Their domain is R and they will be used by the output entity *display* to draw the main area of the watch.

All the internal entities will have N_ϵ as time structure.

The entity *main_mode* will encode the mode of the watch. Its possible values are $\{tim, dat, update, alarm1, alarm2, updalarm1, updalarm2, chime, stopwatch\}$. It transduction is $(main\_mode, \{main\_mode, a, b, c, d, 2s, 2min, update\_mode, updal1\_mode, updal2\_mode\}, \epsilon, (0, tim), f)$. The function f of 9 variables $m, a, b, c, d, 2s, 2min, upm, um1, um2$ ($m, upm, um1, um2$ are the abbreviations for *main_mode*, *update_mode*, *updal1_mode* and *updal2_mode*) is defined by:

$f(m, a, b, c, d, 2s, 2min, upm, um1, um2) =$

dat	$m = tim \wedge d = 1$
tim	$m = dat \wedge d = 1$
tim	$m \notin \{tim, update\} \wedge 2min = 0$
$update$	$m = tim \wedge c = 1 \wedge 2s = 0$
tim	$m = update \wedge b = 1$
tim	$m = update \wedge upm = mode \wedge c = 1$
$alarm1$	$m = tim \wedge a = 1$
$updalarm1$	$m = alarm1 \wedge c = 1$
$alarm1$	$m = updalarm1 \wedge b = 1$
$alarm1$	$m = updalarm1 \wedge um1 = min \wedge c = 1$
$alarm2$	$m = alarm1 \wedge a = 1$
$updalarm2$	$m = alarm2 \wedge c = 1$
$alarm2$	$m = updalarm2 \wedge b = 1$
$alarm2$	$m = updalarm2 \wedge um2 = min \wedge c = 1$
$chime$	$m = alarm2 \wedge a = 1$
$stopwatch$	$m = chime \wedge a = 1$
tim	$m = stopwatch \wedge a = 1$
m	otherwise

The notation:

$$f(...) = \begin{array}{ll} v1 & c1 \\ v2 & c2 \\ ... & ... \end{array}$$

means that

$$f(...) = \begin{array}{lll} if\ c1 & then\ v1 & \\ & elseif\ c2 & then\ v2 \\ & & elseif\ ...then\ ... \end{array}$$

The entity $2min$ is used to implement the time-out that forces to return in the time mode. Its transduction is $(2min, \{main\_mode, a, d, 2min\}, \epsilon, (0, 0), g)$. The function g is defined by:

$$g(m, a, d, 2min) = \begin{array}{ll} 120 - \epsilon & m = tim \wedge d = 1 \\ 120 - \epsilon & m \in \{tim, alarm1, alarm2, chime\} \wedge a = 1 \\ 2min - \epsilon & 2min > 0 \\ 2min & \text{otherwise} \end{array}$$

The entity $2s$ is almost similar. Its transduction is $(2s, \{main\_mode, c, 2s\}, \epsilon, (0,2), h)$ where h is:

$$h(m,c,2s) = \begin{array}{ll} 2s - \epsilon & 2s < 2 \wedge c = 1 \wedge m = tim \\ 2 - \epsilon & m = tim \wedge c = 1 \\ 2 & \text{otherwise} \end{array}$$

The entity *update_mode* encodes what unit is updated by an event d. Its values are $\{none, second, min, 10min, hour, month, dat, day, year, mode\}$. Its transduction is $(update\_mode, \{main\_mode, b, c, update\_mode\}, \epsilon, (0, none), k)$. The function k is defined by:

$$k(m,b,c,upm) = \begin{array}{ll} none & m \neq update \\ none & b = 1 \\ second & m = update \wedge upm = none \\ min & upm = second \wedge c = 1 \\ 10min & upm = min \wedge c = 1 \\ hour & upm = 10min \wedge c = 1 \\ month & upm = hour \wedge c = 1 \\ dat & upm = month \wedge c = 1 \\ day & upm = dat \wedge c = 1 \\ year & upm = day \wedge c = 1 \\ mode & upm = year \wedge c = 1 \\ upm & \text{otherwise} \end{array}$$

The value of the entity *time* is the current time calculated by the watch. Its transduction is $(time, \{d, update\_mode, time\}, \epsilon, (0,0), l)$ where l is the following function:

$$l(d,upm,time) = \begin{array}{ll} mod + (time, 1, 60) & upm = second \wedge d = 1 \\ mod + (time, 60, 600) & upm = min \wedge d = 1 \\ mod + (time, 600, 3600) & upm = 10min \wedge d = 1 \\ mod + (time, 3600, 86400) & upm = hour \wedge d = 1 \\ mod(time + \epsilon, 86400) & \text{otherwise} \end{array}$$

The function *mod* is the usual modulo function and $mod+(x,a,b) = b*div(x,b)+mod(x+a,b)$ where *div* is the integer division.

The entity *date* is similar to *time* but encodes the *date*. We suppose we have functions *add_day*, *add_week_day*, *add_month*, *add_year*, *add_a_day* that given a date return a new date incremented by a day, a day of the week, a month, an year or a day and a day of the week. The transduction of *date* is $(date, \{d, update\_mode, time, date\}, \epsilon, (0,0), l1)$ with:

$$l1(d,upm,time,date) = \begin{array}{ll} add\_day(date) & upm = dat \wedge d = 1 \\ add\_week\_day(date) & upm = day \wedge d = 1 \\ add\_month(date) & upm = month \wedge d = 1 \\ add\_year(date) & upm = year \wedge d = 1 \\ add\_a\_day(date) & time = 86400 - \epsilon \\ date & \text{otherwise} \end{array}$$

The entity *time_mode* keeps track of how to display the time. Its value are $\{24h, am/pm\}$. Its transduction is $(time\_mode, \{d, update\_mode, time\_mode\}, \epsilon, (0, 24h), l2)$ with:

$$l2(d, upm, tm) = \begin{array}{ll} 24h & upm = mode \wedge d = 1 \wedge tm = am/pm \\ am/pm & upm = mode \wedge d = 1 \wedge tm = 24h \\ tm & \text{otherwise} \end{array}$$

The entity *alarm1_stat* is very simple. Its transduction is $(alarm1\_stat, \{main\_mode, d, alarm1\_stat\}, \epsilon, (0, disabled), l3)$ with:

$$l3(m, d, al1s) = \begin{array}{ll} enabled & m = alarm1 \wedge d = 1 \wedge al1s = disabled \\ disabled & m = alarm1 \wedge d = 1 \wedge al1s = enabled \\ al1s & \text{otherwise} \end{array}$$

The entities *alarm2_stat* and *chime_stat* are similar except that they are for the second alarm and chime mode.

The entity *updal1_mode* is quite identical to *update_mode* but simpler. Its values are $\{none, hour, 10min, min\}$, transduction is $(updal1\_mode, \{main\_mode, c, updal1\_mode\}, \epsilon, (0, none), l4)$ with:

$$l4(m, c, updal1m) = \begin{array}{ll} hour & m = alarm1 \wedge c = 1 \\ none & updal1m = min \wedge c = 1 \\ none & m \neq updalarm1 \\ 10min & updal1m = hour \wedge c = 1 \\ min & updal1m = 10min \wedge c = 1 \\ updal1m & \text{otherwise} \end{array}$$

The entity *updal2_mode* is identical to *updal1_mode* for the second alarm.

The entity *alarm1_time* encodes the time of the first alarm and has transduction $(alarm1\_time, \{updal1\_mode, d, alarm1\_time\}, \epsilon, (0, 0), l5)$ with:

$$l5(um1, d, al1t) = \begin{array}{ll} mod + (al1t, 3600, 86400) & um1 = hour \wedge d = 1 \\ mod + (al1t, 600, 3600) & um1 = 10min \wedge d = 1 \\ mod + (al1t, 60, 600) & um1 = min \wedge d = 1 \\ al1t & \text{otherwise} \end{array}$$

The entity *alarm2_time* is similar to *alarm1_time*.

The entity *chrono_mode* indicates whether or not the chronograph is running and has value in $\{run, off\}$. Its transduction is $(chrono\_mode, \{main\_mode, b, d, chrono\_mode\}, \epsilon, (0, off), l6)$ with:

$$l6(m, b, d, chrm) = \begin{array}{ll} run & m = stopwatch \wedge b = 1 \wedge chrm = off \\ off & m = stopwatch \wedge b = 1 \wedge chrm = run \\ off & m = stopwatch \wedge d = 1 \\ chrono\_mode & \text{otherwise} \end{array}$$

The entity *chrono* keeps track of the chronograph's time. Its transduction is ($chrono$, $\{main\_mode, d, chrono\_mode, chrono\}, \epsilon, (0,0), l7$) with:

$$l7(m, d, chrm, chrono) = \begin{array}{ll} 0 & m = stopwatch \wedge d = 1 \\ chrono + \epsilon & m = stopwatch \wedge b = 1 \wedge chrm = off \\ chrono & m = stopwatch \wedge b = 1 \wedge chrm = run \\ chrono & chrm = off \\ chrono + \epsilon & chrm = run \end{array}$$

The entity $30s$ indicates how long the beeper must beep. Its transduction is ($30s$, $\{time$, $alarm1\_time$, $alarm2\_time$, $alarm1\_stat$, $alarm2\_stat$, a, b, c, d,$30s\}$, ϵ, $(0,0), l8$) with:

$l8(time, al1t, al2t, al1s, al2s, a, b, c, d, 30s) =$

$$\begin{array}{ll} 30 & time = al1t \wedge al1s = enabled \\ 30 & time = al2t \wedge al2s = enabled \\ 0 & a = 1 \vee b = 1 \vee c = 1 \vee d = 1 \\ 30s - \epsilon & 30s > 0 \\ 0 & \text{otherwise} \end{array}$$

We can now present the output entities. Four of them are quite similar and rather simple: $d\_power, al1, al2$ and *chime* display in the four small areas of the watch. The transduction of d_power is (d_power, $\{power\}$, 0, 0, $\{(on, good), (weak, weak), (dead, none)\}$). The transduction of $al1$ is $(al1, \{power, alarm1\_stat\}, 0, 0, l9)$ with:

$$l9(power, al1s) = \begin{array}{ll} none & power = dead \\ on & al1s = enabled \\ off & als1 = disabled \end{array}$$

The transduction of $al2$ and *chime* are almost similar and are omitted.

The entity *light* is *on* if the light of the watch has been switched on. Its transduction is $(light, 0, \{power, b\}, \emptyset, o)$ with:

$$o(power, b) = \begin{array}{ll} off & power = dead \\ on & b = 1 \\ off & \text{otherwise} \end{array}$$

The entity *beeper* uses $30s$ to beep at the right instant. Its transduction is ($beeper$, $\{power, time, 30s, chime\_stat\}, 0, 0, p$) with:

$$p(power, time, al1s, al2s, 30s, chs) = \begin{array}{ll} silent & power = dead \\ beep & chs = on \wedge mod(time, 3600) \in [0, 2] \\ beep & 30s \neq 0 \\ silent & \text{otherwise} \end{array}$$

The last output entity *display* presents on the main area of the watch the appropriate data. Its transduction is $(display, \{power, main\_mode, time, date, update\_mode, alarm1\_time, alarm2\_time, chrono, time\_mode\}, 0, 0, q)$ with:

$q(power, m, time, date, upm, al1t, al2t, chrono, tm) =$

$none$	$power = dead$
$date$	$m = dat$
$date$	$upm \in \{month, date, day, year\}$
$al1t$	$m \in \{alarm1, updalarm1\}$
$al2t$	$m \in \{alarm2, updalarm2\}$
$chrono$	$m = stopwatch$
$display24h(time)$	$tm = 24h$
$displayam/pm(time)$	$tm = am/pm$

This temporal automaton implements the digital watch described at the beginning of this section.

Appendix C

Proposition 6 *: A deterministic one-tape Turing machine (abbreviated DTM) can be mapped onto a temporal automaton.*

As a proof of proposition 6 we will present a temporal automaton which performs the same computation as a one-tape deterministic Turing machine.

Let us take a DTM with the following features:

1. A finite set Γ of tape symbols, including a subset $\Sigma \subset \Gamma$ of input symbols and a distinguished blank symbol $b \in \Gamma \setminus \Sigma$;
2. A finite set Q of states, including a distinguished start-state q_0 and two distinguished halt-states q_y and q_n;
3. a transition function δ: $(Q \setminus \{q_y, q_n\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$.

We will construct a temporal automaton whose input is a string $x \in \Sigma^*$ and whose output will be $o \in \{yes, no\}$ if the DTM will reach states q_y or q_n (it will be equally possible to output the result of the computation of DTM on x).

Let N be the time structure of all entities of the temporal automaton M.

M has one input i whose domain D_i is $\Sigma \cup \{end, reset\}$

M has 6 internal entities:

- s_c whose domain is $D_{s_c} = \{read, compute\}$
- s_{tape} whose domain is $D_{s_{tape}} = \Gamma^*$
- s_n whose domain is $D_{s_n} = Z$
- s_{inf} whose domain is $D_{s_{inf}} = Z$
- s_{sup} whose domain is $D_{s_{sup}} = Z$
- s whose domain is $D_s = Q$

M has one output o whose domain is $D_o = \{\perp, yes, no\}$.

For all the internal and output entities the delay δ will be the same and equal to 1.

The transduction of o is from s and its init value is $\perp$.

$$v_o(t) = \begin{cases} yes & \text{if } v_s(t-1) = q_y \\ no & \text{if } v_s(t-1) = q_n \\ \perp & \text{otherwise} \end{cases}$$

The transduction of s_c is from i and its init value is $read$.

$$v_{s_c}(t) = \begin{cases} compute & \text{if } v_i(t-1) = end \\ read & \text{if } v_i(t-1) \in \Sigma \cup \{reset\} \end{cases}$$

This entity s_c keeps the state of M and allows to alternate between computation and reading a string of input.

Let us now bring in some notations to specify the other entities more easily. At first a few functions on Γ^* (set of the finite lists of elements $\in \Gamma$):

f	$: N \times \Gamma^* \to \Gamma$	$f(n,l)$ return the n^{th} element of l
$add-$	$: \Gamma \times \Gamma^* \to \Gamma$	$add-(e,l)$ return the list with head e and tail l
$add+$	$: \Gamma \times \Gamma^* \to \Gamma$	$add+(e,l)$ return the list with last element e and beginning by l
$repl$	$: N \times \Gamma \times \Gamma^* \to \Gamma$	$repl(n,e,l)$ return a list l' identical to l except that the n^{th} element of l' is e.

Now a few abbreviations:

$i = v_i(t-1),\ tape = v_{s_{tape}}(t-1),\ k = v_{s_n}(t-1) - v_{s_{inf}}(t-1) + 1,$

$s_c = v_{s_c}(t-1)$

$\delta_1 = [\delta(v_s(t-1), f(k, tape))]_1$

$\delta_2 = [\delta(v_s(t-1), f(k, tape))]_2$

$\delta_3 = [\delta(v_s(t-1), f(k, tape))]_3$

where $[\ ]_i$ return the i^{th} element of a tuple.

At last a few notations for conditions:

$C_1 : (s_c = compute) \wedge (v_s(t-1) \notin \{q_y, q_n\})$

$C_2 : (\delta_3 = 1) \wedge (v_{s_n}(t-1) = v_{s_{sup}}(t-1))$

$C_3 : (\delta_3 = -1) \wedge (v_{s_n}(t-1) = v_{s_{inf}}(t-1))$
The init value of s_n is 1 and its transduction is:

$$v_{s_n}(t) = \begin{cases} 1 & \text{if } s_c = read \text{ or } i = reset \\ v_{s_n}(t-1) + \delta_3 & \text{if } s_c = compute \text{ and } i \neq reset \end{cases}$$

The init value of s_{inf} is 1 and its transduction is :

$$v_{s_{inf}}(t) = \begin{cases} 1 & \text{if } (s_c = read) \vee (i = reset) \\ v_{s_{inf}}(t-1) - 1 & \text{if } (s_c = compute) \wedge (i \neq reset) \wedge C_3 \\ v_{s_{inf}}(t-1) & \text{otherwise} \end{cases}$$

The init value of s_{sup} is 0 and its transduction is :

$$v_{s_{sup}}(t) = \begin{cases} 0 & \text{if } i = reset \\ v_{s_{sup}}(t-1) + 1 & \text{if } (s_c = read) \wedge (i \neq reset) \\ v_{s_{sup}}(t-1) + 1 & \text{if } (s_c = compute) \wedge (i \neq reset) \wedge C_2 \\ v_{s_{sup}}(t-1) & \text{otherwise} \end{cases}$$

At each time s_n represents the position of the reading head of the DTM, s_{inf} the position of the first element of s_{tape} and s_{sup} the position of the last element of s_{tape} on the infinite tape of the DTM.

The init value of s_{tape} is () the empty list and its transduction is:

$$v_{s_{tape}}(t) = \begin{cases} () & \text{if } i = reset \\ add+(i, tape) & \text{if } (s_c = read) \wedge (i \neq reset) \\ add+(b, repl(k, \delta_2, tape)) & \text{if } C_1 \wedge (i \neq reset) \wedge C_2 \\ add-(b, repl(k, \delta_2, tape)) & \text{if } C_1 \wedge (i \neq reset) \wedge C_3 \\ repl(k, \delta_2, tape) & \text{if } C_1 \wedge (i \neq reset) \wedge \neg C_2 \wedge \neg C_3 \\ tape & \text{otherwise} \end{cases}$$

The init value of s is q_0 and its transduction is:

$$v_s(t) = \begin{cases} q_0 & \text{if } s_c = read \\ \delta_1 & \text{if } C_1 \\ v_s(t-1) & \text{otherwise} \end{cases}$$

The machine M which we have constructed can be used to compute the algorithm of the DTM. For this, put into its input the string x of Σ^* ended by a *stop*. M will at first save the input string and then compute the algorithm of the DTM. If the DTM would end its computation then M give as output whether or not the DTM would accept x as a element of the language that the DTM recognizes. M can also be reset to undertake a new computation by sending to the input a *reset*.□

Contents